Screencast: 27-system-containers.webm or 27-system-containers.mp4

System Containers Background

As mentioned in the virtualization overview, the first form of ***system containers*** available for Linux was **Virtuozzo** made by **SWsoft** (later Parallels, Odin and finally Virtuozzo) back in 2001. Virtuozzo's required GPL components were released as a FLOSS project in 2005 named **OpenVZ**. OpenVZ and Virtuozzo continue but will always be out-of-tree patches (not in the mainline Linux kernel) and as a result don't have much chance for widespread Linux distro inclusion (it is based specifically on RHEL kernels)... but it is widely adopted by hosting providers.

Red Hat mostly cares about *application containers*. Red Hat was using a form of containers (called ***gears***) they devised for their **OpenShift** product, they later switched to Docker and Kubernetes. Red Hat included basic support for LXC in libvirtd/virt-manager but has since discontinued support for **LXC** in RHEL7. Because of disagreements with Docker Inc. regarding their refusal to accept Red Hat produced patches into Docker, as well as not liking several elements of the Docker design ("big fat daemon"), Red Hat decided to make a Docker alternative (not a Docker fork) named **podman**. An early version of podman is available in RHEL7 and a contemporary podman version is included in RHEL8. podman has some innovative features namely *rootless containers* and the ability to also run system containers (with a proper container disk image and execution statement).

Canonical (the company behind the Ubuntu Linux distribution) is the primary developer and maintainer behind **LXC** ("lex see"). For more information, see: https://linuxcontainers.org/lxc/introduction/ They have a management layer on top of LXC named **LXD** ("lex dee"). Canonical also produces daily builds of container images of various Linux distributions.

How are system containers different from application containers?

System containers are like a full Linux OS sans independent kernel... which includes an init system, and standard system services (logging, cron, sshd, etc), user accounts, etc. A system container is basically like a light-weight VM without an independent kernel, the inability to load kernel modules (they must be loaded on the container host), etc. system containers typically have persistent storage (pet). Application containers typically have a single application, no init system, no system services, and if done correctly, often non-persistent storage (cattle).

What System Container technology to use?

If I were building a new system container host, I might just choose LXD. If I was more interested about running legacy stuff, OpenVZ Legacy would be the way to go. If I wanted to run KVM VMs in addition to containers, I'd probably run OpenVZ 7. I have to wonder what OpenVZ 8 (based on the fairly fresh RHEL 8 kernel) will be like whenever it comes out. I really like running a container host as a KVM VM. OpenVZ Legacy is easily runnable under KVM but OpenVZ 7, not so much.

I happen to host a number of hobby domains on an OpenVZ Legacy system... and since OpenVZ is a very mature and feature-complete solution, I'll use it to demonstrate system container concepts.

OpenVZ Legacy Components

OpenVZ Legacy consists of the following components made available from the OpenVZ Legacy software repository:

1. vzkernel - the RHEL6 kernel patched with the large, out-of-mainline-tree OpenVZ patch
2. userland tools - vzctl, vzlist, vzquota, ploop and a script for bash-completion that provides for bash tab completion for most OpenVZ keywords and values
3. OS Templates - a .tar.gz|xz file representing a Linux distribution runtime sans kernel typically with a minimal set of packages and the distros native package manager

Installing OpenVZ Legacy on CentOS 6

Please note, you can't install OpenVZ Legacy on your student VM which is CentOS 8 based.

1. Fresh minimal install of CentOS 6 fully updated
2. EPEL repo added for installing bash-completion package
3. Disable SELinux in `/etc/sysconfig/selinux`. Reboot to ensure it is disabled.
4. Add the OpenVZ Legacy repo: `wget http://download.openvz.org/openvz.repo -O /etc/yum.repos.d/openvz.repo`
5. `yum clean all ; yum install vzkernel vzctl vzquota ploop bash-completion`. Reboot system to use the newly installed OpenVZ kernel
6. Download any desired OS Templates from https://download.openvz.org/template/precreated/ and place them in `/vz/template/cache/`.

Creating an OpenVZ container

I will demonstrate the following creation command in class and in the screencast video:

```
vzctl create 1017 \
   --name centos7 \
   --hostname centos7.montanalinux.org \
   --ipadd 23.88.97.17 \
   --ostemplate centos-7-x86_64 \
   --config vswap-1g \
   --diskspace 10G
```

That will basically create `/vz/private/1017/root.hdd/` directory with a config file and a disk image file. OpenVZ will then mount the disk image on `/vz/root/1017/` and extract `/vz/template/cache/centos-7-x86_64.tar.xz` under the mount point as well as create a new config named `/etc/vz/conf/1017.conf` which is a copy of the specified sample config which in this case is `/etc/vz/conf/ve-vswap-1g.conf`. When the extraction is completed, it'll unmount the disk image.

OpenVZ Legacy Container lifecycle

To list available containers: `vzlist -a`
To start a container: `vzctl start 1017`
To attach to a container: `vzctl enter 1017`
To stop a container: `vzctl stop 1017`
To remove a container: `vzctl destroy 1017`

You can also access the container over the network as long as `openssh-server` is installed and enabled. You can create user accounts and even install a GUI desktop environment (XFCE for example) which you can connect to with a remoting protocol (x2go for example).

You can treat your container just like a server / virtual machine.

Resource Management

OpenVZ has two resource management styles; One is fine grain control with 20 or so settings (see `/proc/userbeancounters` on the container host and/or within the container), and the other (vswap) is much easier to use / understand with two settings, ram and swap. The `vzctl set` parameter is used **DYNAMICALLY** to configure / reconfigure resources. By dynamic I mean you **DO NOT have to restart the container for the resource setting to change / take affect**. Here are the three most common examples using the vswap style:

```
vzctl set 1017 --disksize 20G --save
vzctl set 1017 --ram 2G --save
vzctl set 1017 --swap 1G --save
```

Processes view from within the container and from the container host

Note the output of pstree -nup from within the container and from the container host view. As you can see, a container has its own pid namespace but from the view of the container host, a container is nothing more than a grouping of processes with its own init system (systemd, etc).

```
From within the container:
systemd(1)─┬─systemd-journal(27)
           ├─irqbalance(56)───{gmain}(61)
           ├─alsactl(57)
           ├─dbus-daemon(60,dbus)
           ├─firewalld(63)───{gmain}(213)
           ├─rngd(64)
           ├─x2gocleansessio(81)
           ├─systemd-logind(86)
           ├─systemd-network(104,systemd-network)
           ├─NetworkManager(105)─┬─{gmain}(123)
           │                     ├─{gdbus}(125)
           │                     └─dhclient(227)
           ├─systemd-resolve(127,systemd-resolve)
           ├─atd(144)
           ├─agetty(148)
           ├─agetty(149)
           ├─agetty(150)
           ├─agetty(151)
           ├─agetty(152)
           ├─polkitd(192,polkitd)─┬─{gmain}(209)
           │                      ├─{gdbus}(210)
           │                      ├─{JS GC Helper}(216)
           │                      ├─{JS Sour~ Thread}(220)
           │                      └─{polkitd}(223)
           ├─crond(1177)
           ├─gssproxy(3697)─┬─{gssproxy}(3698)
           │                ├─{gssproxy}(3699)
           │                ├─{gssproxy}(3700)
           │                ├─{gssproxy}(3701)
           │                └─{gssproxy}(3702)
           ├─sshd(5896)
           └─sssd(16352)─┬─sssd_be(16353)
                         └─sssd_nss(16354)
```

From the container host:

```
| (host init, kernel threads, and other host processes)
|
|-systemd(13953)-+-sssd(3635)-+-sssd_be(3636)
|                |            `-sssd_nss(3637)
|                |-anacron(11071)
|                |-systemd-journal(13990)
|                |-irqbalance(14030)---{irqbalance}(14044)
|                |-alsactl(14031)
|                |-dbus-daemon(14034,dbus)
|                |-firewalld(14055)---{firewalld}(14284)
|                |-rngd(14056)
|                |-x2gocleansessio(14086)
|                |-systemd-logind(14100)
|                |-systemd-network(14152,systemd-network)
|                |-NetworkManager(14153)-+-{NetworkManager}(14175)
|                |                       |-{NetworkManager}(14177)
|                |                       `-dhclient(14307)
|                |-systemd-resolve(14179,systemd-resolve)
|                |-atd(14196)
|                |-agetty(14200)
|                |-agetty(14201)
|                |-agetty(14202)
|                |-agetty(14203)
|                |-agetty(14204)
|                |-polkitd(14255,lightdm)-+-{polkitd}(14280)
|                |                        |-{polkitd}(14281)
|                |                        |-{polkitd}(14287)
|                |                        |-{polkitd}(14291)
|                |                        `-{polkitd}(14294)
|                |-crond(16040)
|                |-gssproxy(20641)-+-{gssproxy}(20642)
|                |                 |-{gssproxy}(20643)
|                |                 |-{gssproxy}(20644)
|                |                 |-{gssproxy}(20645)
|                |                 `-{gssproxy}(20646)
|                `-sshd(24040)
|
| (other containers with their process trees)
```

Why are system containers better than full blown VMs?

Since containers use less resources, you can fit more of them on a physical host. As an experiment, I created 1,000 containers on a physical server that had 32GB of RAM. For details see my somewhat dated blog post on the subject. It is hard to imagine running more than a few dozen VMs on a host with 32GB of RAM. There are certain use cases where VMs are better suited... you can run different OSes (not just Linux), you can run different kernels, you can load modules and use kernel-specific features on a per VM basis. Containers just don't offer that level of flexibility. Pluses and minuses.